2.3.1 Algoritma Brute Force

Algoritma brute force untuk membentuk garis didasarkan pada persamaan (2-6), yaitu :

1. Tentukan dua titik ujung (x_1,y_1) dan (x_2,y_2)
2. Jika $x_1 = x_2$ (garis vertikal), maka
   (a) $y = y + 1$ dan x tetap
   (b) gambar titik (x,y) di layar
   (c) Selesai

3. Jika $y_1 = y_2$ (garis horisontal), maka
   (a) $x = x + 1$ dan y tetap
   (b) gambar titik (x,y) di layar
   (c) Selesai

{anggap $x_2 > x_1$, (jika sebaliknya, gantilah x_2 dengan x_1)}

4. Hitung kemiringan garis $m = (y_2 - y_1)/(x_2 - x_1)$
5. $N = x_2 - x_1 + 1$
6. $x = x_1$
7. Ulang sebanyak N kali:
   (a) $y = m(x - x_1) + y_1$
   (b) lakukan pembulatan y_a = Round(y),
   (c) gambar titik (x,y_a) di layar
   (d) $x = x + 1$
8. selesai

Contoh 2.2

Diketahui 2 buah titik A(2,1) dan titik B(8,5) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma *Brute Force*.

Jawab:

1. titik ujung $(x_1,y_1) = (2,1)$ dan $(x_2,y_2) = (8,5)$
2. tidak dipenuhi

3. tidak dipenuhi
4. $m = (5 – 1)/(8 – 2) = 0,67$
5. N = 8 – 2 + 1 = 7
6. untuk $x = 2$ $y = 0,67 (x – 2) + 1$
7. Ulang sebanyak 7 kali

   iterasi ke-1:

   $x = 2$; $y = 0,67. (2 – 2) + 1 = 1$

   pembulatan: $y = 1$

   gambar titik (2,1) di layar

   $x = 2 + 1 = 3$

   iterasi ke-2:

   $x = 3$; $y = 0,67. (3 – 2) + 1 = 1,67$

   pembulatan: $y = 2$

   gambar titik (3,2) di layar .

   $x = 3 + 1 = 4$

   iterasi ke-3:

   $x = 4$; $y = 0,67. (4 – 2) + 1 = 2,34$

   pembulatan: $y = 2$

   gambar titik (4,2) di layar .

   $x = 4 + 1 = 5$

   iterasi ke-4:

   $x = 5$; $y = 0,67. (5 – 2) + 1 = 3,01$

   pembulatan: $y = 3$

   gambar titik (5,3) di layar .

   $x = 5 + 1 = 6$

   iterasi ke-5:

   $x = 6$; $y = 0,67. (3 – 2) + 1 = 3,68$

pembulatan: $y = 4$

gambar titik (6,4) di layar .

$x = 6 + 1 = 7$

iterasi ke-6:

$x = 7$; $y = 0{,}67. (7 – 2) + 1 = 4{,}35$

pembulatan: $y = 4$

gambar titik (7,4) di layar .

$x = 7 + 1 = 8$

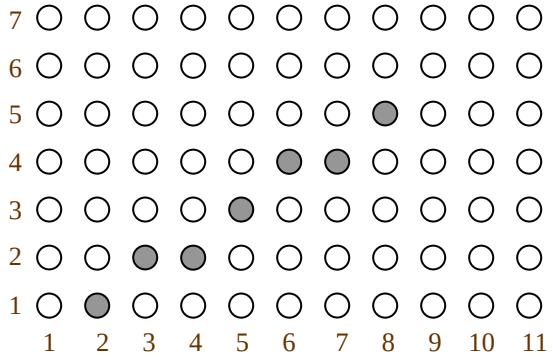
iterasi ke-7:

$x = 8$; $y = 0{,}67. (7 – 2) + 1 = 5{,}02$

pembulatan: $y = 5$

gambar titik (8,5) di layar .

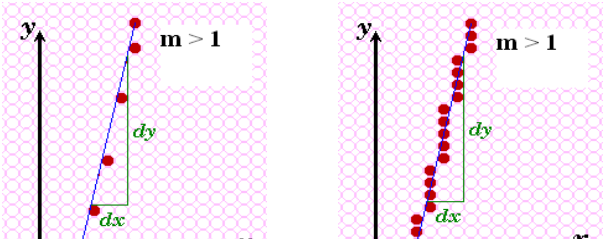
diperoleh titik-titik pembentuk garis : (2,1), (3,2), (4,2), (5,3), (6,4), (7,4), (8,5).



Gambar 2.4: Titik-titik pembentuk garis hasil perhitungan menggunakan algoritma *Brute Force* digambar pada *raster graphics.*

Masalah :

Untuk kemiringan ***m*** > 1, garis menjadi tidak kontinyu (Gambar 2.5(a)) yang mengakibatkan terjadinya *gap* antar piksel, sehingga diperlukan interpolasi (Gambar 2.5(b)).



Gambar 2.5: (a) Garis dengan kemiringan ***m*** > 1, tampak bahwa garis tidak kontinyu (b) setelah dilakukan interpolasi garis menjadi kontinyu.

Selain menggunakan interpolasi, untuk kemiringan garis ***m*** > 1, tukarlah x dengan y maka sudah tidak terjadi *gap* antara titik yang satu dengan yang lain. Sehingga algoritma pembentukan garis untuk ***m*** > 1 adalah sebagai berikut :

1. Tentukan dua titik ujung (x_1,y_1) dan (x_2,y_2)
2. Jika $x_1 = x_2$ (garis vertikal), maka
   (a) $y = y + 1$ dan x tetap
   (b) gambar titik (x,y) di layar
   (c) Selesai

3. Jika $y_1 = y_2$ (garis horisontal), maka
   (a) $x = x + 1$ dan y tetap
   (b) gambar titik (x,y) di layar
   (c) Selesai

{anggap $y_2 > y_1$, (jika sebaliknya, gantilah y_2 dengan y_1)}

4. Hitung kemiringan garis $m = (x_2 - x_1) /(y_2 - y_1)$
5. $N = y_2 - y_1 + 1$
6. $y = y_1$
7. Ulang sebanyak N kali:
   (a) $x = m(y - y_1) + x_1$
   (b) lakukan pembulatan x_a = Round(x),
   (c) gambar titik (x_a,y) di layar
   (d) $y = y + 1$

8. selesai

Contoh 2.3

Diketahui 2 buah titik A(4,3) dan titik B(7,8) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma *Brute Force*.

Jawab:

1. titik ujung A(4,3) dan B(7,8)
2. tidak dipenuhi
3. tidak dipenuhi
4. $m = (7 – 4)/(8 – 3) = 0,6$
5. $N = 8 – 3 + 1 = 6$
6. untuk $y = 3$ dan $x = 0,6. (y – 3) + 4$
7. Ulang sebanyak 6 kali

   iterasi ke-1:

   $y= 3$; $x = 0,6. (3 – 3) + 4 =4$

   pembulatan: $x = 4$

   gambar titik (4,3) di layar .

   $y = 3 + 1 = 4$

   iterasi ke-2:

   $y= 4$; $x = 0,6. (4 – 3) + 4 =4,6$

   pembulatan: $x = 5$

   gambar titik (5,4) di layar .

   $y = 4 + 1 = 5$

   iterasi ke-3:

   $y= 5$; $x = 0,6. (5 – 3) + 4 =5,2$

   pembulatan: $x = 5$

   gambar titik (5,5) di layar .

$y = 5 + 1 = 6$

iterasi ke-4:

$y = 6$; $x = 0{,}6. (6 – 3) + 4 = 5{,}8$

pembulatan: $x = 6$

gambar titik (6,6) di layar .

$y = 6 + 1 = 7$

iterasi ke-5:

$y = 7$; $x = 0{,}6. (7 – 3) + 4 = 6{,}4$

pembulatan: $x = 6$

gambar titik (6,7) di layar .

$y = 7 + 1 = 8$

iterasi ke-6:

$y = 8$; $x = 0{,}6. (8 – 3) + 4 = 7$

pembulatan: $x = 7$

gambar titik (7,8) di layar .

$y = 8 + 1 = 9$

titik-titik pembentuk garis : (4,3), (5,4), (5,5), (6,6), (6,7), (7,8).

Kelemahan algoritma *Brute Force* :

Algoritma ini terlalu lambat karena: pada setiap iterasi terdapat perkalian bilangan pecahan (*floating point*), penjumlahan bilangan pecahan, dan proses pembulatan angka pecahan menjadi bulat (integer).

2.3.2 Algoritma DDA (*Digital Diferential Analyzer*)

DDA adalah algoritma pembentuk garis yang didasarkan pada perasamaan (2-8). Garis dibuat menggunakan titik awal (x_1, y_1) dan titik akhir (x_2, y_2). Setiap koordinat titik (x_k, y_k) yang membentuk garis diperoleh dari perhitungan, kemudian hasil perhitungan dikonversikan menjadi nilai integer. Algoritma ini bisa digunakan untuk menghitung garis dengan semua kemiringan. {

$0 < m < 1$; $m>1$; $-1 < m < 0$; $m < -1$ }. Berikut adalah langkah-langkah pembentukan garis berdasarkan algoritma DDA:

==========

1. Tentukan dua titik yang akan dihubungkan dalam pembentukan garis.
2. Tentukan salah satunya sebagai titik awal (x_1, y_1) dan yang lain sebagai titik akhir (x_2, y_2).
3. Hitung : $dx = x_2 - x_1$ dan $dy = y_2 - y_1$
4. Tentukan *step*, dengan ketentuan berikut:
   - bila $|dx| > |dy|$ maka $step = |dx|$
   - bila tidak, maka $step = |dy|$
5. Hitung penambahan koordinat piksel dengan persamaan:

   $x\_inc = dx / step$

   $y\_inc = dy / step$
6. Koordinat selanjutnya :

   $x = x + x\_inc$ $y = y + y\_inc$
7. Lakukan pembulatan u = Round(x), v = Round(x), kemudian plot piksel (u, v) pada layar
8. Ulangi point 6 dan 7 untuk menentukan posisi piksel berikutnya sampai $x = x_2$ dan $y = y_2$.

Contoh 2.4

Diketahui 2 buah titik A(2,1) dan titik B(8,5) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma DDA.

Jawab:

Titik awal (x_1, y_1) = A(2,1) dan Titik akhir (x_2, y_2) = B(8,5)

$dx = x_2 - x_1 = 8 - 2 = 6$ dan $dy = y_2 - y_1 = 5 - 1 = 4$

Karena: $|dx| > |dy|$, maka $step = |dx| = 6$

$x\_inc = dx / step = 6/6 = 1$

$y\_inc = dy / step = 4/6 = 0{,}67$

==========

Iterasi ke-1: $(x,y) = (2,1)$

$x+x\_inc = 2 + 1 = 3$

$y+y\_inc = 1 + 0{,}67 = 1{,}67$

Koordinat selanjutnya : $(x,y) = (3;\ 1{,}67)$

Pembulatan $(3;\ 1{,}67) \approx (3,2)$. Gambar titik (3,2) dilayar

==========

Iterasi ke-2: $(x,y) = (3;\ 1{,}67)$

$x+x\_inc = 3 + 1 = 4$

$y+y\_inc = 1{,}67 + 0{,}67 = 2{,}34$

Koordinat selanjutnya : $(x,y) = (4;\ 2{,}34)$

Pembulatan $(4;\ 2{,}34) \approx (4,2)$. Gambar titik (4,2) dilayar

==========

Iterasi ke-3: $(x,y) = (4;\ 2{,}34)$

$x_A+x\_inc = 4 + 1 = 5$

$y_A+y\_inc = 2{,}34 + 0{,}67 = 3{,}01$

Koordinat selanjutnya : $(x,y) = (5;\ 3{,}01)$

Pembulatan $(5;\ 3{,}01) \approx (5,3)$. Gambar titik (5,3) dilayar

==========

Iterasi ke-4: $(x,y) = (5;\ 3{,}01)$

$x_A+x\_inc = 5 + 1 = 6$

$y_A+y\_inc = 3{,}01 + 0{,}67 = 3{,}68$

Koordinat selanjutnya : $(x,y) = (6;\ 3{,}68)$

Pembulatan $(6;\ 3{,}68) \approx (6,4)$. Gambar titik (6,4) dilayar

==========

Iterasi ke-5: $(x,y) = (6;\ 3{,}68)$

$x_A+x\_inc = 6 + 1 = 7$

$y_A+y\_inc = 3{,}68 + 0{,}67 = 4{,}35$

Koordinat selanjutnya : $(x,y) = (7;\ 4{,}35)$

Pembulatan $(7;\ 4{,}35) \approx (7,4)$. Gambar titik (7,4) dilayar

==========

Iterasi ke-6: $(x,y) = (7;\ 4{,}35)$

$x_A+x\_inc = 7 + 1 = 8$

$y_A+y\_inc = 4{,}35 + 0{,}67 = 5{,}02$

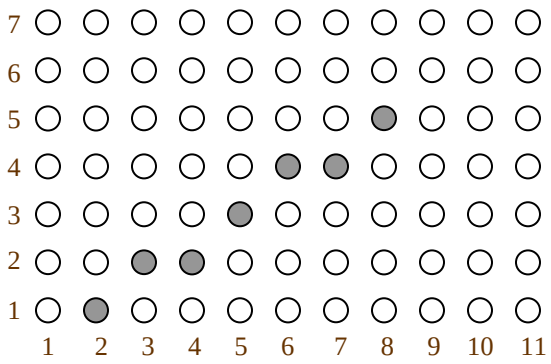
Koordinat selanjutnya : $(x,y) = (8;\ \ 5{,}02)$

Pembulatan $(8;\ \ 5{,}02) \approx (8,5)$. Gambar titik (8,5) dilayar

Karena $x = x_2 = 8$, maka iterasi dihentikan, sehingga diperoleh titik-titik pembentuk garis sebagai berikut: (2,1), (3,2), (4,2), (5,3), (6,4), (7,4) dan (8,5).

==========

Bila digambar pada *raster graphics* diperoleh gambar 2.6:



Gambar 2.6: Titik-titik pembentuk garis hasil perhitungan menggunakan algoritma DDA digambar pada *raster graphics.*

Kelebihan Algoritma DDA dibanding Algoritma *Brute Force*

Algoritma DDA lebih cepat dibanding dengan algoritma *Brute Force* dan baik digunakan untuk kemiringan garis $\boldsymbol{m} > 1$.

Kelemahan Algoritma DDA

- Prosedur untuk menggambar garis masih menggunakan fungsi pembulatan *x* maupun *y*, sehingga memerlukan waktu.
- variabel *x*, *y* maupun *m* memerlukan bilangan real karena kemiringan merupakan nilai pecahan.

2.3.3 Algoritma Bressenham

1. Tentukan dua titik yang akan dihubungkan dalam pembentukan garis.
2. Tentukan salah satu sebagai titik awal (x_0, y_0) dan titik akhir (x_1, y_1).

3. Hitung dx, dy, $2|dy|$ dan $2|dy| - 2|dx|$
4. Hitung parameter : $po = 2|dy| - |dx|$
5. Untuk setiap x_k sepanjang jalur garis, dimulai dengan k = 0
   - bila $p_k < 0$ maka titik selanjutnya adalah:

     (x_k+1, y_k) dan $p_{k+1} = p_k + 2|dy|$
   - bila tidak, titik selanjutnya adalah:

     (x_k+1, y_k+1) dan $p_{k+1} = p_k + 2|dy| - 2|dx|$
6. Ulangi nomor 5 untuk menentukan posisi piksel berikutnya, sampai

   $x = x_1$ dan $y = y_1$.

Contoh 2.5

Diketahui 2 buah titik A(2,1) dan titik B(8,5) bila titik A sebagai titik awal dan titik B sebagai titik akhir, maka buatlah garis yang menghubungkan titik tersebut dengan menggunakan algoritma Bressenham.

Jawab:

Titik awal (x_0, y_0) = A(2,1) dan Titik akhir (x_1, y_1) = B(8,5)

$dx = x_1 - x_0 = 8 - 2 = 6$ dan $dy = y_1 - y_0 = 5 - 1 = 4$

$m = dy/dx = 4/6$; kemiringan garis berada diantara 0 dan 1 : $0 < m < 1$

$2|dx| = 2.6 = 12$; $2|dy| = 2.4 = 8$ dan $2|dy| - 2|dx| = 8 - 12 = -4$

$p_o = 2|dy| - |dx| = 8 - 6 = 2$

==========

Iterasi ke-1 (k = 0):

Titik awal = (2,1)

$Po = 2 > 0$, maka titik selanjutnya adalah

$x = 2 + 1 = 3$ dan $y = 1 + 1 = 2$, koordinat selanjutnya : (3,2)

$p_1 = p_0 + 2|dy| - 2|dx| = 2 - 4 = -2$

==========

Iterasi ke-2 (k = 1):

Titik awal = (3,2)

$P_1 = -2 < 0$, maka titik selanjutnya adalah

$x = 3 + 1 = 4$ dan $y = 2$, koordinat selanjutnya : (4,2)

$p_2 = p_1 + 2|dy| = -2 + 8 = 6$

==========

Iterasi ke-3 (k = 2):

Titik awal = (4,2)

$P_2 = 6 > 0$, maka titik selanjutnya adalah

$x = 4 + 1 = 5$ dan $y = 2 + 1 = 3$, koordinat selanjutnya : (5,3)

$p_3 = p_2 + 2|dy| - 2|dx| = 6 - 4 = 2$

==========

Iterasi ke-4 (k = 3):

Titik awal = (5,3)

$P_3 = 2 > 0$, maka titik selanjutnya adalah

$x = 5 + 1 = 6$ dan $y = 3 + 1 = 4$, koordinat selanjutnya : (6,4)

$p_4 = p_3 + 2|dy| - 2|dx| = 2 - 4 = -2$

==========

Iterasi ke-5 (k = 4):

Titik awal = (6,4)

$P_3 = -2 < 0$, maka titik selanjutnya adalah

$x = 6 + 1 = 7$ dan $y = 4$, koordinat selanjutnya : (7,4)

$p_5 = p_4 + 2|dy| = -2 + 8 = 6$

==========

Iterasi ke-6 (k = 5):

Titik awal = (7,4)

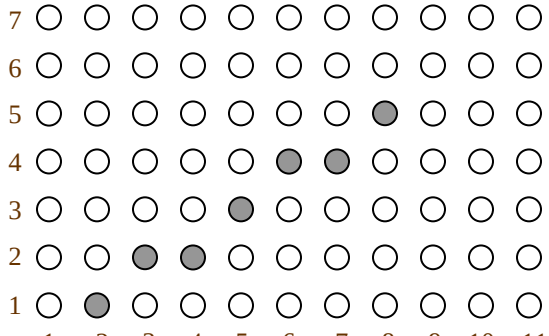
$P_5 = 6 > 0$, maka titik selanjutnya adalah

$x = 7 + 1 = 8$ dan $y = 4 + 1 = 5$, koordinat selanjutnya : (8,5)

==========

Karena $x = x_2 = 8$, maka iterasi dihentikan, sehingga diperoleh titik-titik penyusun garis sebagai berikut: (2,1), (3,2), (4,2), (5,3), (6,4), (7,4) dan (8,5).

Bila digambar pada *raster graphics* diperoleh Gambar 2.8:



Gambar 2.8: Titik-titik pembentuk garis dengan kemiringan 0 < m < 1, hasil perhitungan menggunakan algoritma Bressenham yang digambar pada *raster graphics.*

2.4 Lingkaran

2.4.1 Simetris Delapan Titik

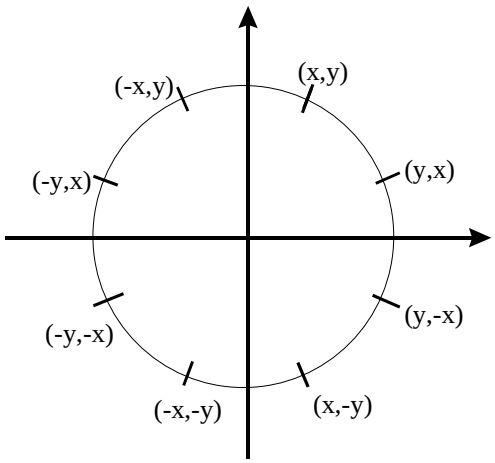
Pembuatan kurva lingkaran dapat dilakukan dengan menentukan titik awal (*x*,*y*) yang terletak pada lingkaran, maka tujuh titik yang lain (yang terletak pada lingkaran juga) dapat ditentukan sebagai berikut :

$$(-x,y),\ (x,-y),\ (-x,-y),\ (y,x),\ (-y,x),\ (y,-x),\ (-y,-x)$$

Sehingga terbentuk delapan titik:

$$(x,y),\ (-x,y),\ (x,-y),\ (-x,-y),\ (y,x),\ (-y,x),\ (y,-x),\ (-y,-x)$$

Dengan demikian sebenarnya hanya diperlukan untuk menghitung segmen 45° dalam menentukan lingkaran selengkapnya.



Gambar 2.11: Delapan titik simetris pada lingkaran

2.4.2 **Algoritma *Midpoint***

Dari sini diperoleh langkah-langkah algoritma pembentuk lingkaran.

- Langkah-langkah Algoritma Lingkaran Midpoint adalah:
  1. Tentukan jari-jari r dan pusat lingkaran (x_p, y_p), kemudian setting sedemikian rupa sehingga titik awal berada pada: $(x_0, y_0) = (0 , r)$
  2. Hitung nilai parameter :

     $p_0 = \frac{5}{4} - r$ Jika jari-jari r pecahan

     $p_0 = 1 - r$ Jika jari-jari r bulat

  3. Untuk setiap posisi x_k, dimulai dengan k = 0 berlaku ketentuan:
     - bila $p_k < 0$ maka titik selanjutnya adalah (x_k+1, y_k) dan $p_{k+1} = p_k + 2\, x_{k+1} + 1$
     - bila tidak, titik selanjutnya adalah $(x_k+1, y_k - 1)$ dan $p_{k+1} = p_k + 2\, x_{k+1} + 1 - 2\, y_{k+1}$

  4. Tentukan titik simetris pada ketujuh oktan yang lain
  5. Gerakkan setiap posisi piksel (x, y) pada garis lingkaran dengan titik pusat (x_p, y_p) dan plot nilai koordinat : $x = x + x_p$, $y = y + y_p$
  6. Ulangi langkah 3 sampai dengan 5 hingga $x \geq y$

Contoh 2.9

Buatlah gambar kurva lingkaran dengan pusat lingkaran (4,6) dan jari-jari 8, perhitungan berdasarkan dari oktan kuadran pertama dimana $x = 0$ sampai $x = y$. Koordinat titik awal dimulai dari $(x,r) = (0,8)$. Karena jari-jari r bulat, maka gunakan $P_0 = 1 - r$.

Iterasi ke-1:

K = 0 X_0 = **0** Y_0 = r = **8** $P_0 = 1 - r = 1 - 8 = -7$

Karena $P_0 < 0$, maka :

$X_1 = X_0 + 1 = 0 + 1 = 1$ dan $Y_1 = Y_0 = 8$, jadi Titik selanjutnya : (1,8)

$P_1 = p_0 + 2x_1 + 1 = -7 + 2.(1) + 1 = -4$

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(1,8), (–1,8), (1, –8), (–1, –8), (8,1), (–8,1), (8, –1), (–8, –1)

Gerakkan setiap posisi piksel (*x, y*) pada garis lingkaran dengan titik pusat (4,6) diperoleh titik-titik berikut

(5, 14), (3, 14), (5, –2), (3, –2), (12, 7), (–4, 7), (12, 5), (–4, 5)

Iterasi ke-2:

K = 1 X_1 = 1 Y_1 = 8 P_1 = – 4

Karena P1 < 0, maka

$X_2 = X_1 + 1 = 1 + 1 = 2$ dan $Y_2 = Y_1 = 8$, jadi Titik selanjutnya : (2,8)

$P_2 = p_1 + 2x_2 + 1 = -4 + 2.(2) + 1 = 1$

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(2,8), (–2,8), (2, –8), (–2, –8), (8,2), (–8,2), (8, –2), (–8, –2)

Gerakkan setiap posisi piksel (*x, y*) pada garis lingkaran dengan titik pusat (4,6) diperoleh titik-titik berikut

(6, 14), (2, 14), (6, –2), (2, –2), (12, 8), (–4, 8), (12, 4), (–4, 4)

Iterasi ke-3:

K = 2 X_2 = 2 Y_2 = 8 P_2 = 1

Karena $P_2 > 0$, maka

$X_3 = X_2 + 1 = 2 + 1 = 3$ dan $Y_3 = Y_2 - 1 = 8 - 1 = 7$, jadi Titik selanjutnya : (3,7)

$P_3 = p_2 + 2x_3 + 1 - 2y_3 = 1 + 2.(3) + 1 - 2.(7) = -6$

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(3,7), (–3,7), (3, –7), (–3, –7), (7,3), (–7,3), (7, –3), (–7, –3)

Gerakkan setiap posisi piksel (*x, y*) pada garis lingkaran dengan titik pusat (4,6) diperoleh titik-titik berikut

(7, 13), (1, 13), (7, –1), (1, –1), (11, 9), (–3, 9), (11, 3), (–3, 3)

Iterasi ke-4:

K = 3 $X_3 = 3$ $Y_3 = 7$ $P_3 = -6$

Karena $P_3 < 0$, maka

$X_4 = X_3 + 1 = 3 + 1 = 4$ dan $Y_4 = Y_3 = 7$, jadi Titik selanjutnya : (4,7)

$P_4 = p_3 + 2x_4 + 1 = -6 + 2.(4) + 1 = 3$

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(4,7), (–4,7), (4, –7), (–4, –7), (7,4), (–7,4), (7, –4), (–7, –4)

Gerakkan setiap posisi piksel (*x, y*) pada garis lingkaran dengan titik pusat (4,6) diperoleh titik-titik berikut

(8, 13), (0, 13), (8, –1), (0, –1), (11, 10), (–3, 10), (11, 2), (–3, 2)

Iterasi ke-5:

K = 4 $X_4 = 4$ $Y_4 = 7$ $P_4 = 3$

Karena $P_4 > 0$, maka

$X_5 = X_4 + 1 = 4 + 1 = 5$ dan $Y_5 = Y_4 - 1 = 7 - 1 = 6$, jadi Titik selanjutnya : (5,6)

$P_5 = p_4 + 2x_4 + 1 - 2y_4 = 3 + 2.(5) + 1 - 2.(6) = 2$

Dengan algoritma simetris delapan titik, maka diperoleh titik-titik berikut :

(5,6), (–5,6), (5, –6), (–5, –6), (6,5), (–6,5), (6, –5), (–6, –5)

Gerakkan setiap posisi piksel (*x, y*) pada garis lingkaran dengan titik pusat (4,6) diperoleh titik-titik berikut

(9, 12), (–1, 12), (9, 0), (–1, 0), (10, 11), (–2, 11), (10, 1), (–2, 1)

Iterasi ke-6:

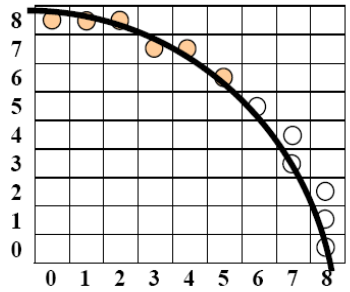
K = 5 $X_5 = 5$ $Y_5 = 6$ $P_5 = 2$

Karena $P_5 > 0$, maka

$X_6 = X_5 + 1 = 5 + 1 = 6$ dan $Y_6 = Y_5 - 1 = 6 - 1 = 5$, jadi Titik selanjutnya : (6,5)

Iterasi dihentikan karena X > Y.

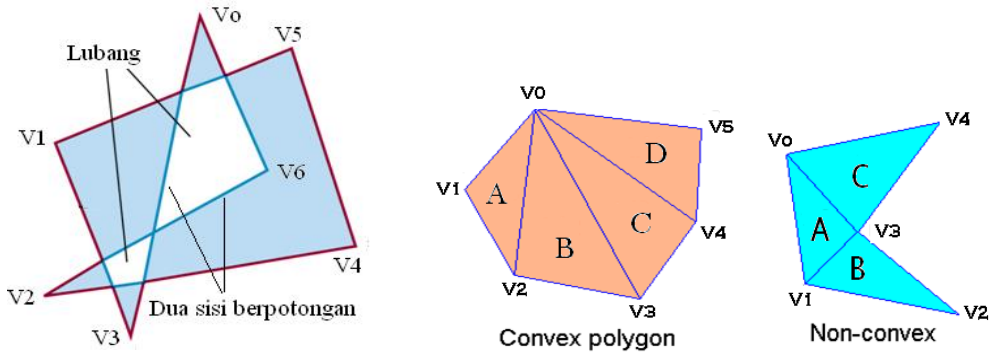
Bila digambar, hasil untuk oktan ke-1 ditunjukkan oleh gambar (2.13).



Gambar 2.13: Posisi piksel pada pembentukan lingkaran dengan titik pusat (0,0) dan jari-jari 8

2.5 Polygon

Polygon adalah kumpulan garis lurus yang saling menyambung hingga membentuk suatu luasan. Garis-garis ini disebut *edge* (sisi polygon). Titik pertemuan tiap dua sisi disebut verteks. Biasanya polygon dinyatakan dengan koordinat verteks-verteks ini. Ada dua jenis polygon yaitu polygon sederhana dan polygon tidak sederhana. Ciri polygon sederhana adalah tidak semua verteks berada pada bidang yang sama, tidak mempunyai sisi-sisi yang berpotongan, dan tidak mempunyai lubang.

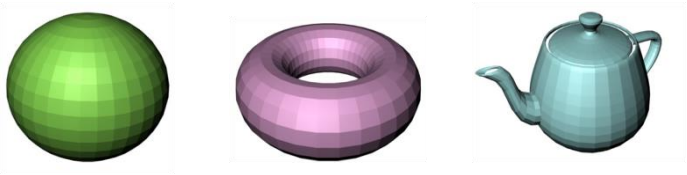


(a) Polygon Tidak sederhana (b) Polygon sederhana

Gambar 2.14 : Polygon sederhana dan polygon tidak sederhana

Polygon sederhana dibagi menjadi dua yaitu convex polygon dan non convex polygon. Ciri convex polygon adalah semua sudut interiornya < 180^o, atau setiap segmen garis yang dihasilkan

dari dua buah verteks sembarang dalam polygon berada didalam polygon. Bentuk polygon yang paling sederhana adalah segitiga, karena semua polygon dapat dipecah-pecah menjadi bagian yang terkecil yaitu segitiga seeprti pada Gambar 2.14(b). Mengapa polygon ? Dengan polygon secara praktek kita bisa melakukan pendekatan untuk membentuk permukaan setiap obyek 3-D, jika kita mempunyai jumlah polygon yang cukup. Sebagai contoh permukaan bola, torus, dan teko seperti pada Gambar 2.15 dapat dibuat dari beberapa polygon.



Gambar 2.15: Permukaan bola, torus dan teko yang terbentuk dari banyak polygon.

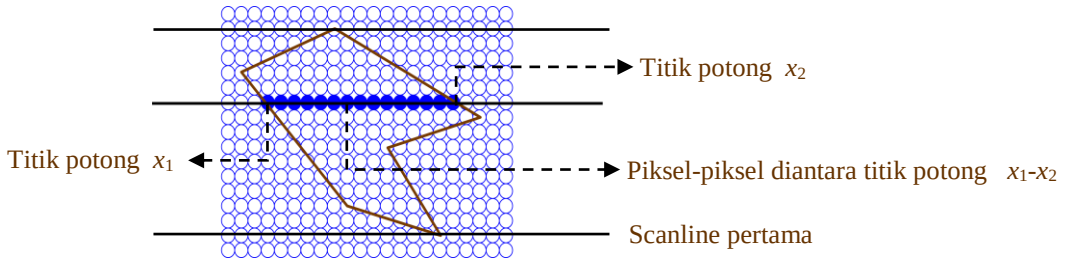
2.6 *Filling Polygon*

Dalam grafis, pemberian warna dibutuhkan untuk mempercantik tampilan polygon. Karena itu diperlukan algoritma khusus untuk mengisi warna pada polygon tersebut. Teknik atau algoritma untuk pengisian warna pada polygon disebut *filling polygon* atau *area filling*. Ada dua macam dasar pendekatan *filling polygon* pada sistem raster yaitu scan-line *Polygon Fill Algorithm* dan *Boundary-Fill Algorithm*:

2.6.1 *Scan Line Polygon Fill Algorithms*

Pemberian warna pada polygon dilakukan dengan cara men-*scan* secara horisontal dari kiri ke kanan. Hal ini dilakukan untuk mendapatkan titik potong dengan tepi polygon, kemudian mengurutkan nilai-nilai titik potong *x* dari kiri ke kanan dan memberi warna pada piksel-piksel diantara dua pasangan berurutan (x_1- x_2). Hal ini dilakukan dari garis scan yang paling bawah (nilai y terkecil) hingga garis scan yang paling atas seperti pada Gambar 2.16. Metode ini bisa juga digunakan untuk pengisian warna pada obyek-obyek sederhana lainnya, misalnya lingkaran, ellip dan lain-lain.

Scanline terakhir

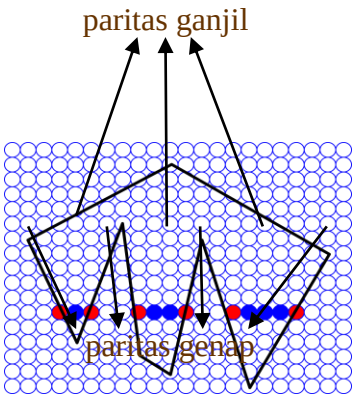


Gambar 2.16: Metode scan-line

Bagaimana kita tahu bahwa piksel tersebut berada didalam polygon atau diluar polygon ?

6.2.1.1 Inside-Outside test

Perhatikan Gambar 2.17. Untuk mengidentifikasi bagian dalam dan bagian luar digunakan aturan paritas ganjil-genap yang biasa disebut sebagai *odd-even rule* atau *odd parity rule*. Semula kita men-set paritas dengan nilai genap. Setiap ditemukan titik potong nilai paritas dibalik, yang semula genap dibalik menjadi ganjil, sebaliknya yang semula ganjil dibalik menjadi genap. Beri warna pada piksel jika paritasnya Ganjil.



Gambar 2.17: aturan paritas ganjil-genap untuk mengisi warna

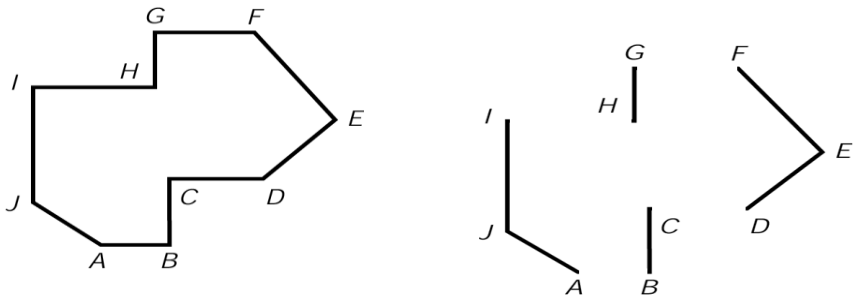
Masalah: Pada Gambar 2.18(a)**,** verteks **a, b, c,** dan **d** merupakan pertemuan dari dua buah segmen garis. Mengapa pada verteks **a** dan **d** dihitung sekali, sedangkan pada verteks **b** dan **c** dihitung dua kali ?

(a) (b)

Gambar 2.18: perhitungan paritas pada verteks **a, b, c,** dan **d.**

Solusi: Buatlah perputaran tepi polygon searah jarum jam seperti pada Gambar 2.18(b). Check, apabila pada suatu verteks arah anak panahnya berubah (pada verteks **b** dan **c** yaitu: naik-turun atau turun-naik) maka pada verteks tersebut dihitung dua kali. Sebaliknya jika arah anak panahnya tidak berubah (pada verteks **a** dan **d** yaitu: naik-naik atau turun-turun) maka pada verteks tersebut dihitung sekali.

Masalah: Perhatikan Gambar 2.19(a), Bagaimana perhitungan tepi horisontal **AB, CD, GF,** dan **HI** ?



(a) (b)

Gambar 2.19: polygon dengan sisi-sisi horisontal

Solusi: abaikan verteks-verteks yang terletak pada tepi horisontal, atau jangan dimasukkan dalam perhitungan paritas ganjil-genap, sehingga tampak seperti pada Gambar 2.19(b).

Algoritma scanline menggunakan kaidah paritas ganjil-genap:

- Tentukan titik potong garis scan dengan semua sisi polygon

- Urutkan titik potong tadi berdasarkan koordinat sumbu *x*
- Warnai semua piksel diantara pasangan titik potong (x_1-x_2) yang terletak didalam polygon menggunakan kaidah paritas ganjil-genap.

Kelemahan algoritma ini adalah : memerlukan biaya tinggi (*big cost*) karena pada setiap sisi polygon selalu dilakukan pengujian terhadap piksel-piksel.

Solusi: menggunakan konsep *edge table* (ET)

2.6.1.2 Edge Table (ET)

edge table (ET) adalah tabel yang berisi sisi-sisi polygon. Kita akan menggunakan dua *edge table* yang berbeda, yaitu: *Active Edge Table* (AET) dan *Global Edge Table* (GET). (AET) digunakan untuk menyimpan semua sisi polygon yang berpotongan dengan garis scan, sedangkan GET digunakan untuk menyimpan semua sisi polygon dan meng-*update* AET.

***Active Edge Table* (AET)**

- Tabel berisi satu entry per sisi yang berpotongan dengan garis scan aktif.
- Pada setiap garis scan yang baru :
  - Hitung titik potong baru untuk semua sisi menggunakan rumus : $x_{i+1} = x_i + \frac{1}{m}$
  - Tambahkan setiap sisi baru yang berpotongan
  - Hapus setiap sisi yang tidak berpotongan
- Untuk efisiensi Update AET, kita tetap menjaga GET.

***Global Edge Table* (GET)**

- Tabel yang berisi informasi tentang semua sisi-sisi polygon
- GET mempunyai satu tempat unttuk setiap garis scan
  - Setiap tempat menyimpan daftar sisi yang mempunyai nilai $y_{\min}$
  - Setiap sisi ditentukan hanya dalam satu tempat

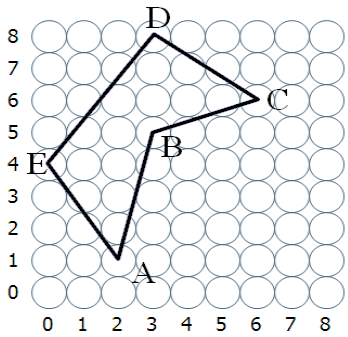
- Tiap-tiap entry dalam GET berisi
  - Nilai y_{max} dari sisi
  - Nilai $x_{@ymin}$ (nilai x pada titik y_{min})
  - Nilai pertambahan x (1/m)

***Scan Line Polygon Fill Algorithms* menggunakan ET, GET dan AET**

1. Tambahkan sisi-sisi polygon ke GET
2. Set y ke koordinat y terkecil dalam GET
3. Inisialisasi, set AET = kosong
4. Ulang sampai AET dan GET kosong
   a. Tambahkan sisi-sisi dari GET ke AET yang mana $y_{min} = y$.
   b. Hilangkan sisi-sisi dari AET bila $y_{max} = y$.
   c. Urutkan AET berdasarkan x
   d. Warnai piksel yang terletak diantara pasangan titik potong dalam AET
   e. Untuk setiap sisi dalam AET, ganti x dengan $x + 1/m$
   f. Set $y = y + 1$ untuk bergerak ke garis scan berikutnya

Contoh 2.10

Diketahui polygon berikut, gunakan konsep *edge table* untuk mewarnai polygon tersebut.



Masukan GET $(y_{max}, x_{@ymin}, 1/m)$

AB → (5, 2, ¼)

BC → (6, 3, 3)

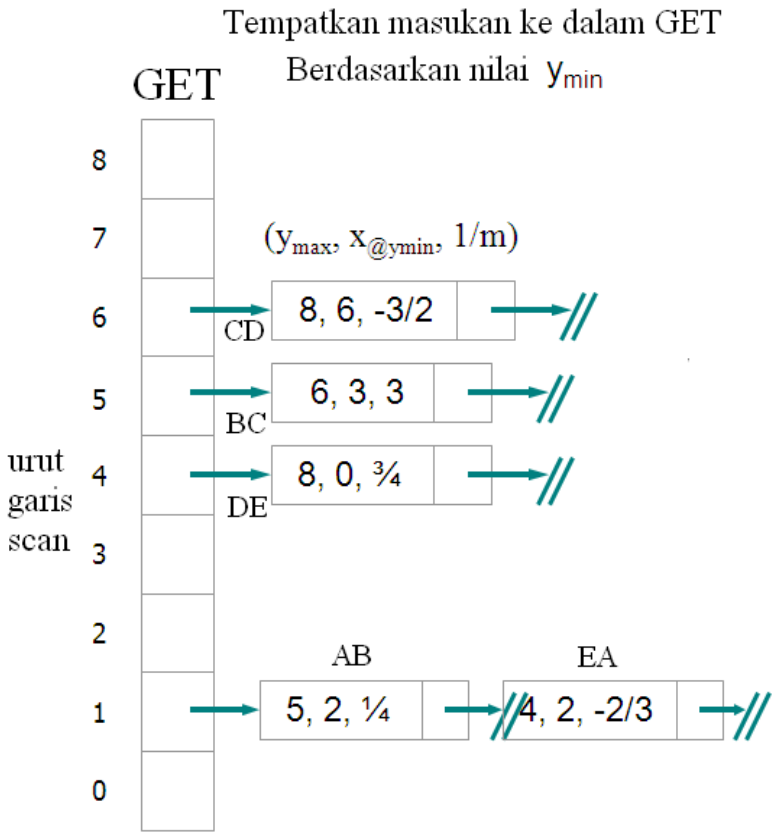
CD → (8, 6, -3/2)

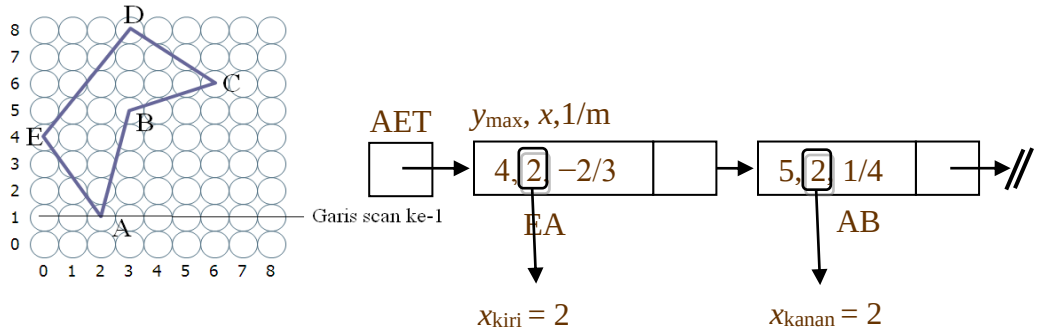
DE → (8, 0, 3/4)

EA → (4, 2, -2/3)

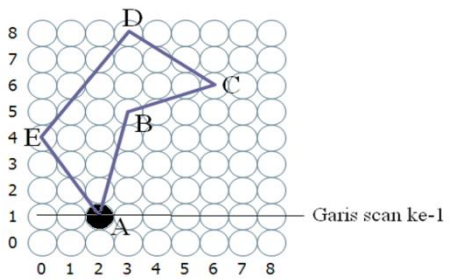
Sisi-sisi pembentuk polygon

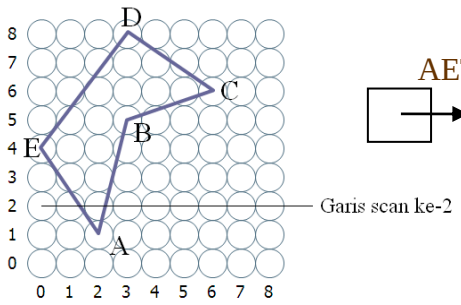
AB = (2, 1), (3, 5)
BC = (3, 5), (6, 6)
CD = (6, 6), (3, 8)
DE = (3, 8), (0, 4)
EA = (0, 4), (2, 1)

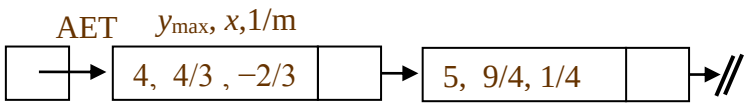




Pewarnaan dilakukan diantara titik potong ($x_{kiri} - x_{kanan}$) = (2 − 2), hasilnya adalah





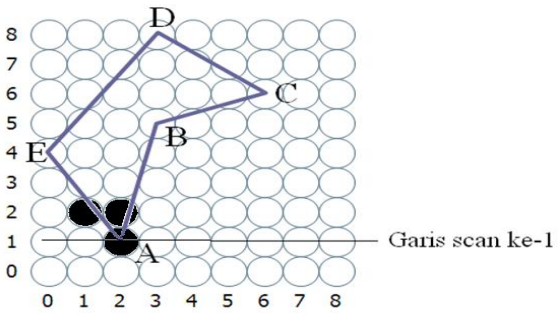


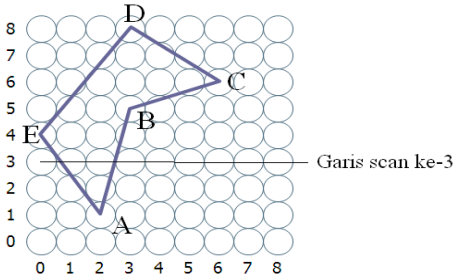
$x_{i+1} = x_i + 1/m$
$= 2 - 2/3$
$= 4/3$

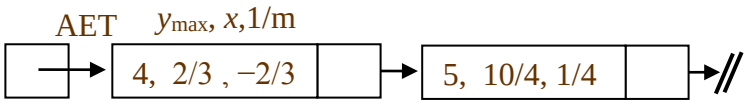
$x_{i+1} = x_i + 1/m$
$= 2 + 1/4$
$= 9/4$

$x_{kiri} = 4/3 \approx 1$ dan $x_{kanan} = 9/4 \approx 2$

Pewarnaan dilakukan diantara titik potong ($x_{kiri} - x_{kanan}$) = (1 − 2), hasilnya adalah





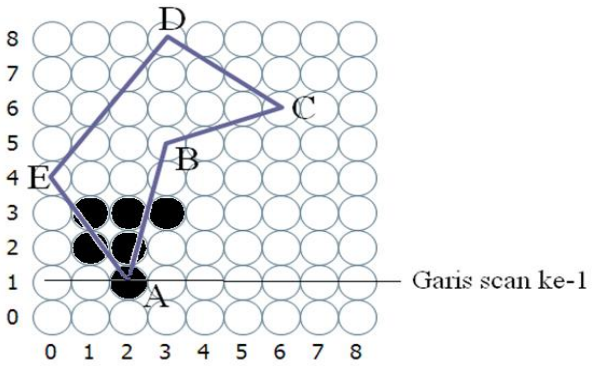


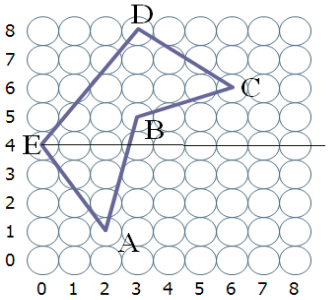
$x_{i+1} = x_i + 1/m$
$= 4/3 - 2/3$
$= 2/3$

$x_{i+1} = x_i + 1/m$
$= 9/4 + 1/4$
$= 10/4$

$x_{kiri} = 2/3 \approx 1$ dan $x_{kanan} = 10/4 \approx 3$

Pewarnaan dilakukan diantara titik potong ($x_{kiri} - x_{kanan}$) = (1 − 3), hasilnya adalah





AET y_{max}, x,1/m

8, 0 , 3/4 → 5, 11/4, 1/4 → //

~~EA~~ = DE

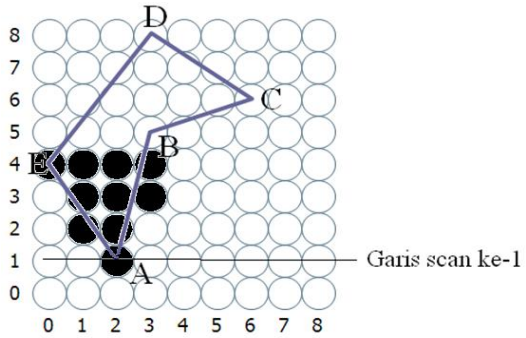
AB

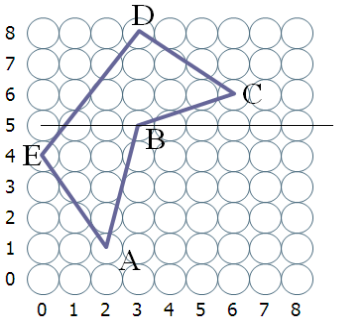
pada EA, y_{max} = 4. EA harus dihapus dari AET. Dalam GET sisi DE tersimpan pada y = 4. Jadi sisi EA diganti dengan sisi DE

$x_{i+1} = x_i + 1/m$
$= 10/4 + 1/4$
$= 11/4$

$x_{kiri} = 0$ dan $x_{kanan} = 11/4 \approx 3$

Pewarnaan dilakukan diantara titik potong $(x_{kiri} - x_{kanan}) = (0 - 3)$, hasilnya adalah





AET y_{max}, x,1/m

8, 3/4 , 3/4 → 6, 3, 3 → //

DE

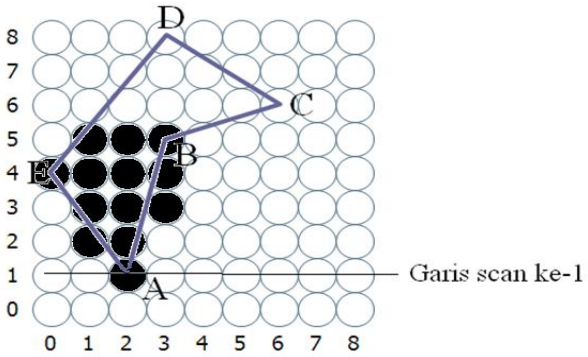
~~AB~~ = BC

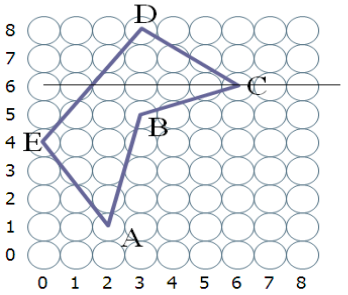
$x_{i+1} = x_i + 1/m$
$= 0 + 3/4$
$= 3/4$

pada AB, y_{max} = 5. AB harus dihapus dari AET. Dalam GET sisi BC tersimpan pada y = 5. Jadi sisi AB diganti dengan sisi BC

$x_{kiri} = 3/4 \approx 1$ dan $x_{kanan} = 3$

Pewarnaan dilakukan diantara titik potong $(x_{kiri} - x_{kanan}) = (1 - 3)$, hasilnya adalah





AET y_{max}, x,1/m

8, 6/4 , 3/4 → 8, 6, −3/2

DE

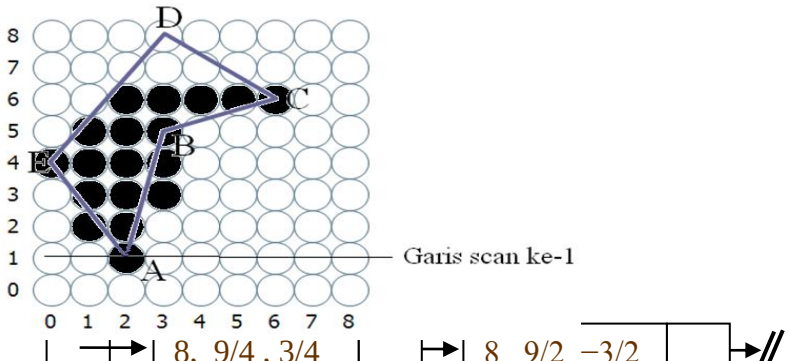
~~BC~~ = CD

$x_{i+1} = x_i + 1/m$
$= 3/4 + 3/4$
$= 6/4$

pada BC, $y_{max} = 6$. BC harus dihapus dari AET. Dalam GET sisi CD tersimpan pada $y = 6$. Jadi sisi BC diganti dengan sisi CD

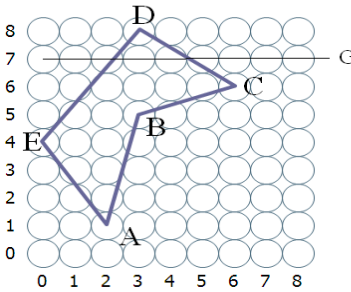
$x_{kiri} = 6/4 \approx 2$ dan $x_{kanan} = 6$

Pewarnaan dilakukan diantara titik potong $(x_{kiri} - x_{kanan}) = (2 - 6)$, hasilnya adalah



8, 9/4 , 3/4 → 8, 9/2, −3/2

DE CD

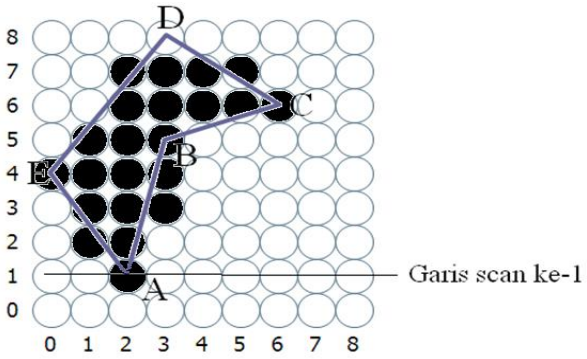


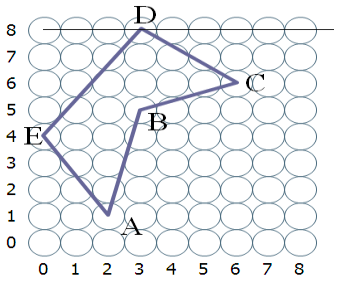
$x_{i+1} = x_i + 1/m$
$= 6/4 + 3/4$
$= 9/4$

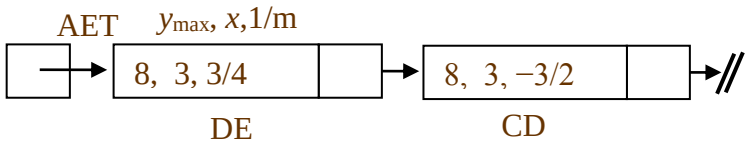
$x_{i+1} = x_i + 1/m$
$= 6 - 3/2$
$= 9/2$

$x_{kiri} = 9/4 \approx 2$ dan $x_{kanan} = 9/2 \approx 5$

Pewarnaan dilakukan diantara titik potong ($x_{kiri} - x_{kanan}$) = (2 − 5), hasilnya adalah





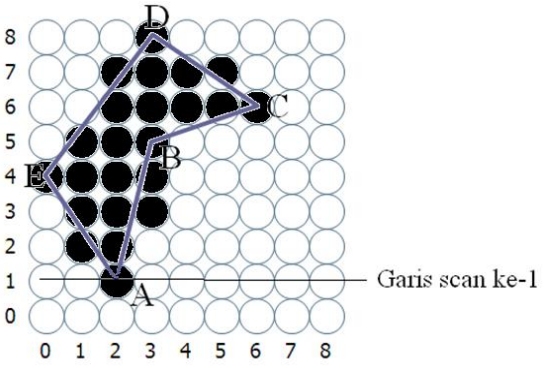


$x_{i+1} = x_i + 1/m$
$= 9/4 + 3/4$
$= 3$

$x_{i+1} = x_i + 1/m$
$= 9/2 - 3/2$
$= 3$

$x_{kiri} = 3$ dan $x_{kanan} = 3$

Pewarnaan dilakukan diantara titik potong ($x_{kiri} - x_{kanan}$) = (3 − 3), hasilnya adalah



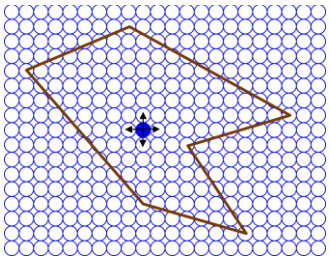
Karena sisi polygon sudah habis, maka dalam GET ataupun AET proses dihentikan.

2.6.2 *Boundary-Fill Algorithm*

Prosedur *boundary-fill* menerima tiga parameter yaitu: koordinat titik (*x*,*y*), warna isi dan warna garis batas. Proses pengisian warna tertentu dimulai dari titik (*x*,*y*), kemudian memeriksa posisi titik tetangganya, apakah titik tetangga tersebut memiliki warna batas:

- Jika tidak, warnai titik tersebut dengan warna tertentu.
- Selanjutnya periksa lagi posisi dan warna titik tetangganya.
- Proses diulangi terus hingga seluruh titik pada area pengisian telah diuji.

Dengan teknik ini pengisian warna dimulai dari sebuah titik yang berada didalam area (*x*,y) polygon dan mewarnai piksel dari titik tetangganya hingga semua piksel yang berada didalam polygon telah diwarnai seperti pada Gambar 2.20.

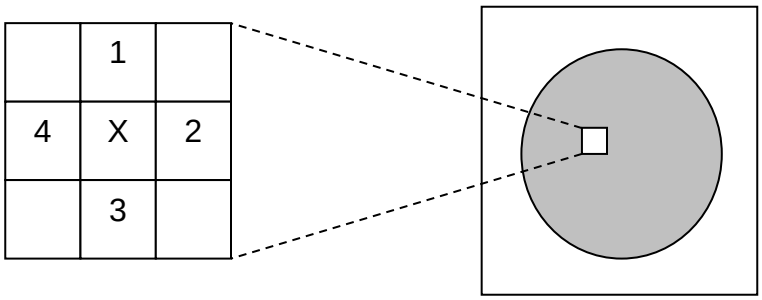


Gambar 2.20: Metode *Boundary-Fill*

Ada 2 macam cara untuk melihat titik tetangga, yaitu :

- 4 tetangga, melihat titik yang berada diatas, bawah, kanan dan kiri
- 8 tetangga, melihat titik yang berada diatas, bawah, kanan, kiri, pojok kiri atas, pojok kiri bawah, pojok kanan atas dan pojok kanan bawah.

a) 4- tetangga



b) 8-tetangga

1	2	3
8	X	4
7	6	5

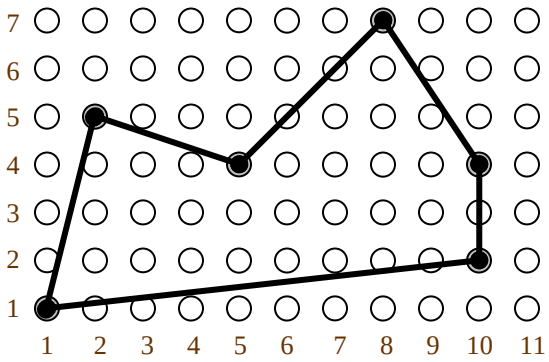
Contoh 2.11

diketahui : polygon = {(1,1), (2,,5), (5,4), (8,7), (10,4), (10,2), (1,1)}, lakukan *Area Filling* menggunakan algoritma *Boundary Fill Algorithm* 4-tetangga.

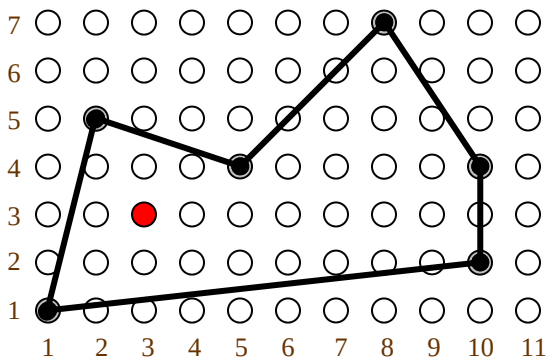
Jawab:

titik-titik sebagai pembentuk polygon = {(1,1), (2,,5), (5,4), (8,7), (10,4), (10,2), (1,1)}.

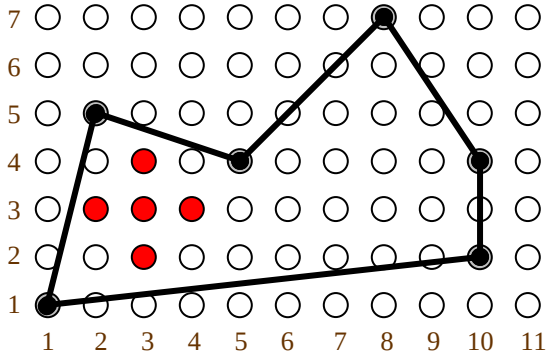
Bila poligon tersebut digambar, diperoleh gambar berikut :



Misalkan titik awal pencarian adalah (3,3). Tandai titik (3,3) dengan warna tertentu, misalnya warna merah. Lihat 4-tetangganya, yaitu titik (3,2), (3,4), (2,3), (4,3).



Ke-4 tetangga tersebut bukan garis batas poligon, sehingga 4-titik tersebut diwarnai merah.



Titik yang telah diproses: (3,3)

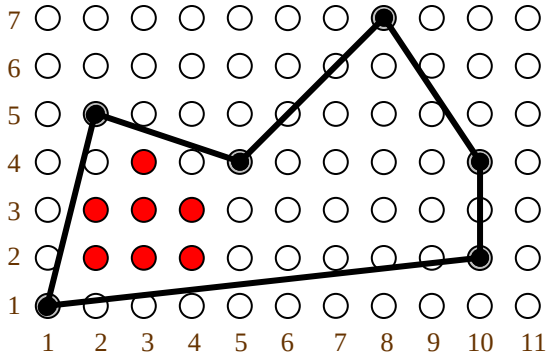
Titik yang belum diproses : (3,2), (3,4), (2,3), (4,3)

Ambil titik (3,2).

Titik yang telah diproses: (3,2), (3,3)

Titik yang belum diproses : (3,4), (2,3), (4,3)

4-tetangga titik tersebut adalah (3,3), (3,1), (2,2), (4,2). Terlihat bahwa titik (4,2) dan (2,2) bukan garis batas poligon, sehingga diwarnai dengan warna merah. Titik (3,3) sudah diwarnai. Titik (3,1) adalah garis batas jadi tidak diwarnai.

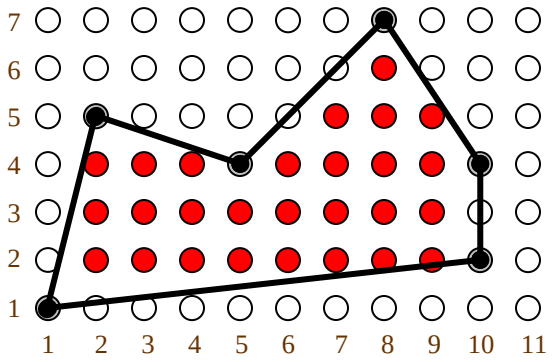


Titik yang telah diproses: (3,3)(3,2)

Titik yang belum diproses : (3,4), (2,3), (4,3) (2,2), (4,2)

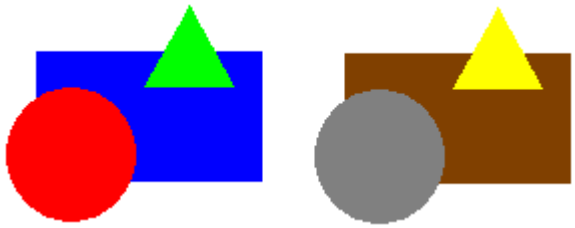
Ambil titik (3,4). 4-tetangga titik tersebut adalah (3,3), (3,5), (2,4), (4,4). Titik (3,3) sudah diwarnai. Titik (3,5), (2,4) dan (4,4) adalah garis batas jadi tidak diwarnai.

Proses diulang sehingga seluruh bagian dalam poligon diwarnai dengan warna merah.



2.6.3 *Flood-Fill Algorithm*

Terkadang kita ingin mewarnai (atau memberi warna yang baru) pada sebuah area yang mempunyai warna lebih dari satu. Perhatikan gambar 2.21 berikut



Gambar 2.21: penggantian warna obyek menggunakan *Flood-Fill Algorithm* . (a) lingkaran berwarna merah, segitiga berwarna hijau dan persegi berwarna biru. Obyek tersebut warnanya diubah menjadi (b) lingkaran berwarna abu-abu, segitiga berwarna kuning dan persegi berwarna coklat

Algoritma ini dimulai dari titik yang berada didalam area (*x*,*y*) dan mengganti semua pikselnya dengan warna baru sehingga bagian dalam area mempunyai warna yang sama. Pengujian titik tetangga bisa menggunakan 4-tetangga atau 8-tetangga.

Soal-Soal Latihan

1. Apa yang dimaksud dengan output primitif ? Sebutkan !

2. Diketahui 2 buah titik A dan titik B. Bila titik A sebagai titik awal dan titik B sebagai titik akhir, tentukan titik-titik antara yang menghubungkan titik A dan titik B sehingga membentuk garis AB dengan menggunakan (a) Algoritma *brute force* (b) algoritma DDA (c) algoritma Bressenham , jika :

   i) A(3,2) dan B(11, 6)

   ii) A(3,2) dan B(7, 7)

   iii) A(–5, 10) dan B(0, 0)

   iv) A(–5, 4) dan B(0,0)

3 Berdasarkan soal no.2, bagaimana menurut saudara, mana algoritma yang lebih baik, Algoritma *brute force* , algoritma DDA atau algoritma Bressenham ? Mengapa ?

4. (a) Buatlah gambar kurva lingkaran dengan pusat lingkaran (0,0) dan jari-jari 6, perhitungan berdasarkan dari oktan kuadran pertama dimana $x = 0$ sampai $y = r$. Koordinat titik awal dimulai dari $(x,r) = (0,6)$. Untuk mempermudah perhitungan gunakan $P_0 = 1 – r$ (sekali lagi, ini hanya untuk mempermudah perhitungan dalam contoh). (b) sama seperti soal (a), tetapi pusat lingkarang di P(2,5).

5. Diketahui : polygon = {(2,1), (3,6), (5,4), (8,8), (10,4), (12,2), (2,1)}, lakukan *Area Filling* menggunakan (a) algoritma *Scan Line Polygon* (b) algoritma *Boundary Fill*.